

ngTyping Test

A typing test written in Angular.IO and demonstrated in Electron.

Overview

This application presents a demonstration of the use of TypeScript, Angular.IO, and Electron. The use of these three technologies together allows a developer to rapidly prototype and build out an application that can run on various OSes by harnessing the power of NodeJS and Electron. Semi-native packages can be created for Windows, MacOS, and Linux distributions with simple commands.

This demo shows the application running in an Electron environment. The Electron environment is nothing more than a specially constructed version of a browser window, specifically one built around Chromium, that runs the "web-application" locally. With considerations during the development of the application it could also be hosted on a standard web-server. This gives the solution an added level of flexibility, those desiring an application that can run offline have an option for the Electron hosted version while thin-clients have the option of using the hosted version. The development and deployment of this type of application benefits from the aspect that very little code changes are required across each version.

This documentation will cover the technical level of this demonstration.

Main Project

The root folder for this project contains a number of configuration files for building the project. Focusing on each of these files would require an in-depth discussion of the technology used to operate this project. Instead of covering each file in a detailed manner this documentation will enumerate and briefly describe what their function as opposed to a detailed analysis of the contents of the files.

Configuration Files

File	Description
<code>.angular-cli.json</code>	Used for the configuration of the Angular-CLI. This configuration file is used to describe the project to the Angular-CLI for building, etc.
<code>.editorconfig</code>	This file is used to describe basic IDE editor settings which should allow the code to be stylistically consistent across developer and IDEs.
<code>.gitignore</code>	Files to be ignored by source control are listed here.
<code>.travis.yml</code>	CI build configuration.
<code>electron-builder.json</code>	Configuration for the Electron application and build.
<code> karma.config.js</code>	Unit test configuration.
<code>package-lock.json</code>	Used by NPM to resolve dependencies in a quick format. Helps insure that anyone consuming this project gets the same dependencies.
<code>package.json</code>	Used to describe the project. Basic NPM configuration.
<code>postcss.config.js</code>	Configures..
<code>protractor.config.js</code>	E2E test configuration
<code>tsconfig.json</code>	TypeScript compilation configuration.
<code>tslint.json</code>	Configuration for TSLint (linting for TypeScript).
<code>webpack.config.js</code>	Webpack configuration.

Electron Files

It would be wise to note here that there is a single file for the Electron application included in the root of the project. The `main.ts` file is the base entry point for the Electron application. It is used to configure the events to be handled by the Electron application sub-system and handle creating the window which is displayed.

Other Files and Folders

Each folder will contain a root `README.md` file. These files are handled via the GitHub documentation engine such that if you visit any of the folders via the GitHub website they will be displayed for that folder at the top of the page. This project is intended to do the same. Additional files, `README.doc.md` files will be included in order to provide supplemental technical information about the contents of the folder.

Electron Application

This folder contains all the support files for the Electron/Server-side application. These are actually used at runtime.

Electron App

The Electron app is managed through `app.ts`. This file handles setting up the Electron application, registering the basic events, and loading the main application page.

Constants

Every application will have static string values used for configuration or events names. This module exports a two hashes, one for all the event names and one for configuration keys. By using a hash where the values are a string it allows us to "strongly type" those strings so that they will be insured to be the same in all locations which need to reference them.

Note: For all `README.md` files further down in the directory structure, their root header should be set to two hash signs: `##`

Configuration

Application configuration is handled in several ways. The app can pull configuration from configuration files, configuration tied to a specific user, and through the launch of the application on the CLI.

Root Module

Contains the default export for this "namespace". It is the configuration for the application.

CLI

The `cli.ts` file exports a module to get settings from the CLI and configure them on the configuration object.

Electron

The electron module in configuration contains all the functionality to manage configuration of the Electron application. This is a meta-level configuration. These configurations are applicable to all users.

The module exports methods for loading and saving this configuration.

User

The user module in the configuration section is used for setting up the definition of configurations for specific users. Also contains a default set of configuration values.

Web

The web module provides functions for loading and saving configurations while the application is running from a web server. This is necessary because the manner in which the configuration is saved differs between the two.

Inter-Process Communications

Inter-process communication occurs in Electron when the renderer process, the process hosting the client application, talks to the behind the scenes main process, the Electron process. This process to process communication allows the web app to perform long running tasks outside of the rendering thread. This keeps the UI responsive. It also allows the code to be logically split between a server and client architecture. This means that in this solution a web application can be hosted within the renderer process and with very little change be hosted on a standard application server.

Root

The default export of this "namespace" is a function to register the IPC listeners for each of the individual modules.

Configuration

The configuration module handles all the IPC messaging for configuration related functions. The loading and saving of configuration is handled through this module.

Test

IPC listeners related to the testing functionality. This module includes the listeners for getting test text, saving test results, and retrieving results for a user.

Logging

The Electron app allows for the use of the logging methods on the `console` object. This is not ideal as those messages are only sent to the console. While helpful in debug and development environments, a full fledged logging framework needs to be available for production use.

This project chooses to use the Winston logging tools. These tools are configured via the classes in this folder.

The use of the `index.ts` file and a default export within the file allows a very simplistic import into any application file requiring logging support.

Repository

The *repository* folder contains modules for data repository actions. These modules deal with connecting to a data store and performing CRUD operations.

Root

The default module for this folder contains the basic ability to connect to the data-store. This project uses a data-store which mimics MongoDB. The API for the external module is exactly the same as the MongoDB driver for NodeJS. This allows this example to easily switch over to using MongoDB.

Test-Results (*test-results*)

The *testResults* module contains the operations for working with test result objects. It supports saving/logging a new result to the data store. It also provides functionality for loading a user's results.

User

The *user* module contains the data store operations for user objects. This includes loading, creating, updating.

Test

The *test* folder contains all the modules relating to the **Test** object. This includes the definition for the object structure as well as various other supporting objects.

Result

The `ITestResult` interface defined in `*result.ts` defines the structure of a test result.

Test Text Info

The file *testText.ts* is the module which exports an enumeration and interface for the test info. The enumeration, `TestTextLocation`, details where the test text was pulled. This is useful when introducing more places which could provide test text examples. The `ITestText` interface is used to define the text used for testing the user.

Word

The *word* module provides a utility interface for identifying error'd words within the text.

User

The user folder contains all modules pertaining to the users.

User

The user module simply contains the interface for defining the user object's data structure.

Project Support (Support folder)

The *support* folder contains any utilities and helpers to get the project up and running. These resources are not required for the actual functioning of the application but are intended to facilitate any of the functions for supporting builds, documentation, etc.

TypeScript

All the items within this folder should be TypeScript files. A special task, `build-support`, has been created to compile this directory using the `tsconfig.json` file within the directory.

File Descriptions

File	Description
<code>concat-readme.ts</code>	Used to concatenate the <code>README.md</code> and <code>README.doc.md</code> files into a single markdown file.
<code>create-pdf.ts</code>	Used to take a markdown file and turn it into a PDF document.

doc-utils.ts A set of utility functions used to concatenate the markdown files and produce a TOC.
tsconfig.json Since the files here are support only the way they are compiled is a little different than the rest of the project. This config file lets the TypeScript compilation take place a little differently.

Utilities

These modules export utility functions which are used outside of any server/client aspect.

License Information

Copyright 2018 - Michael Gardner

This software is released under the MIT License:

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Portions of this code base are Copyright 2017 - Maxime GRIS

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.